


TIME TRACKER



Author: Aoibheann Farrell • 11711959

Supervisor: Renaat Verbruggen

Blog: <http://blogs.computing.dcu.ie/wordpress/aoibhinnfarrell/>

ABSTRACT

TimeTracker is a cross-platform timesheet web application designed to allow employees of a company to enter their weekly timesheets simply and quickly. It provides management with a simple method of tracking hours worked by employees, time spent on projects and monitoring of annual leave. TimeTracker was built with C# using ASP.NET MVC, an open source web application framework that applies the Model-View-Controller design pattern to the ASP.NET framework. It allows users to create, update, edit and delete timesheets from any device. Existing timesheets can be searched and viewed. Real time notifications alert connected users when a new timesheet, company or project has been added. Administrative users also have access to interactive charts to view important details such as employees hourly rate, annual leave and project progress.

TABLE OF CONTENTS

Introduction.....	3
Overview.....	3
<i>Motivation</i>	3
<i>Research</i>	4
Design and Development Tools.....	5
Glossary.....	7
 Design and Implemenation.....	 8
Architecture Overview.....	8
Application Architecture	9
Diagrams	11
 Problems	 14
Issue: Data Access	14
Issue: ViewModels being saved to Database	14
Issue: Creating a Single Database Context.....	15
 Testing.....	 15
Unit Testing.....	15
Automated Testing	19
User Validation	20
 Future work.....	 20
Scaling and Performance	20
Data Analysis and Prediction	20
 Appendices	 21

Introduction

Overview

Successfully bringing a project to completion on time and within budget and resource constraints is a universal challenge faced by most businesses. One very important aspect of managing this is ensuring that employee hours are tracked accurately. This allows project progression to be monitored closely and aids with payroll and client billing. This is where the TimeTracker application can help. TimeTracker is a timesheet web application that allows hours worked by the employees of a company to be tracked and monitored. This provides management with the ability to ensure overtime is kept in check, monitor project progress and ensure clients are billed accurately for projects. TimeTracker provides a cross-platform mobile web application that allows for accurate time tracking and efficient project management .

Motivation

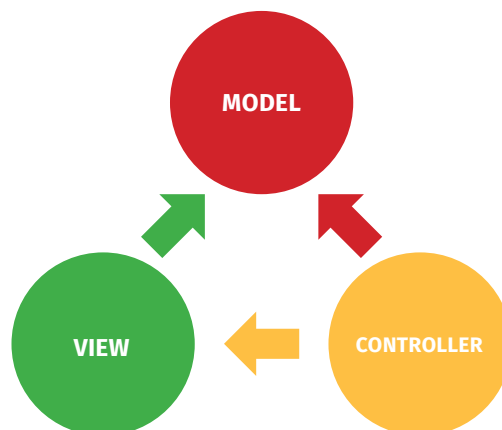
The idea for TimeTracker was developed during my intra placement at Spanish Point Technologies Ltd. Spanish Point Technologies are a software company who provide consulting and build services based on Microsoft technologies. Currently employees use a SharePoint list to enter their weekly timesheet details which is then exported as an excel spreadsheet at the end of the month for analysis by the project manager and financial department. This is not an ideal situation for a number of reasons. For example if an employee does not have access to a computer entering details from a mobile device can be difficult and time consuming as SharePoint does not have strong native functionality for mobile access. This method also does not allow any time or project monitoring functionality to the project manager. This means that it can be difficult to keep clients updated on the progress of a particular project or to monitor if a project is currently within time or budget constraints.

Research

As part of the project-planning phase extensive research was done on best practices for designing an ASP.NET MVC web application and these were then applied in the project.

- Isolation – controllers should be isolated from dependencies such as data access classes and configuration.
- IoC Containers – use IoC containers to manage external dependencies.
- ViewModels – create a ViewModel for POST actions to provide data binding.
- HtmlHelper – use HtmlHelpers to generate view HTML to keep design consistent.
- Domain Logic – separate domain logic from the application.
- Interfaces – define interfaces to deal with data access.

This research also highlighted the main advantage to using the MVC pattern, which is its adherence to the principle of separation of concerns, which means separating the system into distinct components with each component addressing a specific concern. This is a sensible approach to take especially in a web application, as the user interface tends to change much more frequently than the data storage. Having the data storage tightly coupled to the user interface means even minor changes can require a lot of coding. The MVC pattern makes the application more flexible and loosely coupled.



As the above diagram illustrates both the controller and view depend upon the model but the model does not depend upon either the view or controller. This allows the model to be built and tested independently of the visual representation of the data. Another advantage to isolating components in MVC is that it ensures that testing the application a more streamlined process as each component can be tested independently of the others.

DESIGN AND DEVELOPMENT TOOLS

TimeTracker was developed using [C#](#), [HTML5](#), [JavaScript](#) and [Razor Syntax](#). C# was chosen as it is designed for building applications that run on the .NET Framework and its similarity with Java allowed me to start to working productively within a short time frame. User Interface elements were developed using a combination of HTML5, JavaScript and Razor Syntax. Razor Syntax allows server code to be embedded in web pages. This allows dynamic HTML content to be created at runtime.

[Visual Studio Community 2013](#) was the chosen IDE. It is free open source IDE that provides many extensions that made the development process much easier. The version control system that was chosen was [Git](#) a free open-source distributed version control system that integrates easily into Visual Studio. Source code was also pushed to [Github](#), a web based Git repository hosting service. An [SQL Server](#) database was then used for data persistence.

[StructureMap](#), an open source IoC container for the .NET framework, provided dependency injection within the application. A number of third party extensions such as Glimpse and Resharper were used during development and debugging to improve the code quality. [Glimpse](#) is a diagnostics platform that provides real time diagnostics and insights into the application under development. It inspects web requests in real time and provides insights to help with debugging. [Resharper](#) helps to find and fix errors and code smells, navigate and refactor which helps to produce quality code in faster time. StructureMap, Glimpse and Resharper were installed in Visual Studio using the NuGet package manager. NuGet is built into Visual Studio Community and provides a quick and easy way to include third party packages into a project. NuGet copies library files to the solution and automatically updates the projects within the solution by adding references to the packages and changing config files automatically.

[Google Chart Tools](#) were used to provide data visualization to the application. Google provides a free API that allows for the creation of customizable interactive charts using the applications data. It was chosen as it integrates very well with ASP.NET MVC views. The charts are exposed using JavaScript classes in the views and are then rendered to screen using HTML5 and SVG allowing for cross-platform portability a very important aspect of the project. Data is retrieved from the database through controller methods and then formatted in Json, the Json is the passed to the view for rendering.

[SignalR](#) and [Toastr](#) were used in conjunction to create real-time push notifications in the application. SignalR is an ASP.NET library that provides the ability to push content from the server to all connected clients in real time. This is done using [WebSockets](#) a HTML5 API that enables two-way communication between the server and connected clients. SignalR provides an abstraction above the transport layer that is required for real time communication. A SignalR connection starts initially as HTTP, which means if WebSockets is not available it will use other transports to make the connection. Toastr is a JavaScript library that allows for the creation of simple toast pop up notifications.

Glossary

- **ASP.NET** – Is an open source server-side web application framework for web development, developed by Microsoft.
- **ASP.NET Identity** – Membership system for ASP.NET applications.
- **Azure** - Microsoft's cloud based platform.
- **CRUD** - Create, Read, Update and Delete.
- **Entity Framework** - open source object relational mapping framework.
- **HtmlHelpers** – Methods used to modify HTML in ASP.NET MVC.
- **IoC** – Inversion of Control.
- **.NET** - The .NET framework is a development framework from Microsoft that provides a controlled environment where software can be developed, installed and executed on Windows based operation Systems.
- **ORM** – Object Relational Mapping.
- **Razor** - Programming syntax for embedding server code in web pages.
- **SignalR** – ASP.NET library for real-time functionality.
- **StructureMap** – open source IoC container.
- **Toastr** – JavaScript library for toast notifications.
- **Toast Notification** – auto-expiring window of information.

Design and implementation

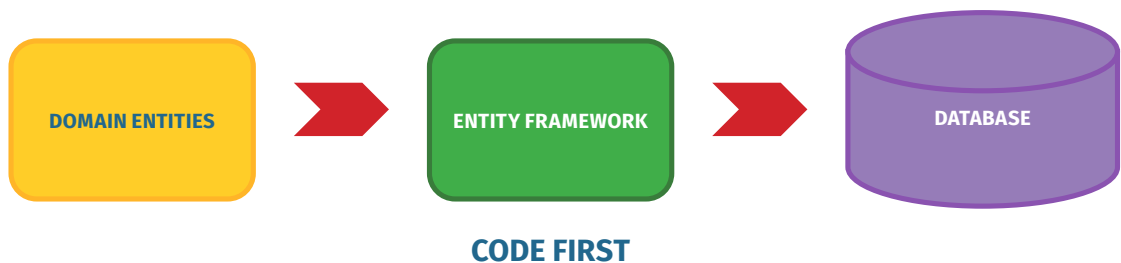
Architecture Overview

TimeTracker is a web application built upon the ASP.NET MVC framework that extends the powerful open source ASP.NET framework. It allows for the development of dynamic web applications using the Model-View-Controller design pattern. This pattern separates the business logic of the application from the user interface.

- **Model** – are responsible for maintaining state within the application, this data is then persisted to the database.
- **View** – are responsible for the UI elements of the application. They render the information to be displayed to the user and pass
- **Controller** – are responsible for manipulating model data, handling user requests and choosing the appropriate view to display to the user.

The **Entity Framework** was used to provide data access, query, insert, update and delete functionality. Entity is an Object Relational Mapping tool that provides an automated mechanism for accessing and storing information in a database. Entity 6 also allows for Code-First design, this means that rather than designing the database first, domain classes that encapsulate the domain logic of the application were created first and the entity framework then generated the database schema based upon these classes and their configuration.

ASP.NET Identity was used to handle membership and authentication within the application. The Identity framework also uses the Entity Framework Code First approach to store users information in the database. This allows for greater control over the schema of user information that is created for users in the system.



Application Architecture

The TimeTracker application is divided into two projects:

- timeTracker.Domain
- timetracker.Web

The timeTracker.Domain project is a class library that holds all the domain objects or entities needed for the application. These domain entities implement the domain logic of the application. They handle the data that is passed between the database and the rest of the application. This library also contains an abstract interface called ITimeTrackerDataSource, which is used to access the domain entities in the class library. The class library creates a Dynamic-Link-Library (DLL) that allows it to be referenced and used in other projects.

The classes contained in timeTracker.Web are;

- Company
- Project
- TimeSheet
- TimeSheetEntry
- ITimeTrackerDataSource

The timeTracker.Web project is the main ASP.NET web application and contains a reference to the timeTracker.Domain project providing access to the domain entities. This project handles all the request handling and user interface elements of the application.

The timeTracker. Web project contains;

- Models
- Configuration files
- Controllers
- Database context
- Views
- Entity migration files
- ViewModels
- Scripts

The timeTracker.Web project contains a class [TimeTrackerDb](#) this is the database context for the application. It is responsible for interacting with the database. TimeTrackerDb inherits from an Entity framework class [IdentityDbContext<ApplicationUser>](#) that is defined in the Entity Framework. It contains base classes for the Identity system. This allows user data and application data to be stored in a single database. TimeTrackerDb contains a DbSet for every entity in the database; DbSet is an entity set that allows for CRUD operations. During runtime the TimeTrackerkerDb class manages the entity objects, populating them with data, tracking any changes made to them and persisting the data back to the database.

TimeTrackerDb also implements the [ITimeTrackerDataSource](#) interface that was defined in the domain project. This abstraction allows the controllers to be decoupled from the database context making the system more flexible, maintainable and testable.

Diagrams

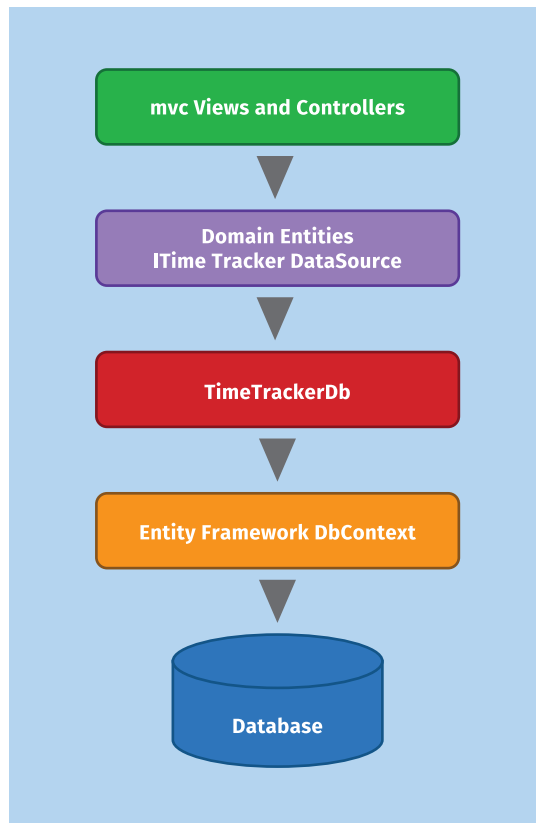


Figure 1: TimeTracker Architecture Overview

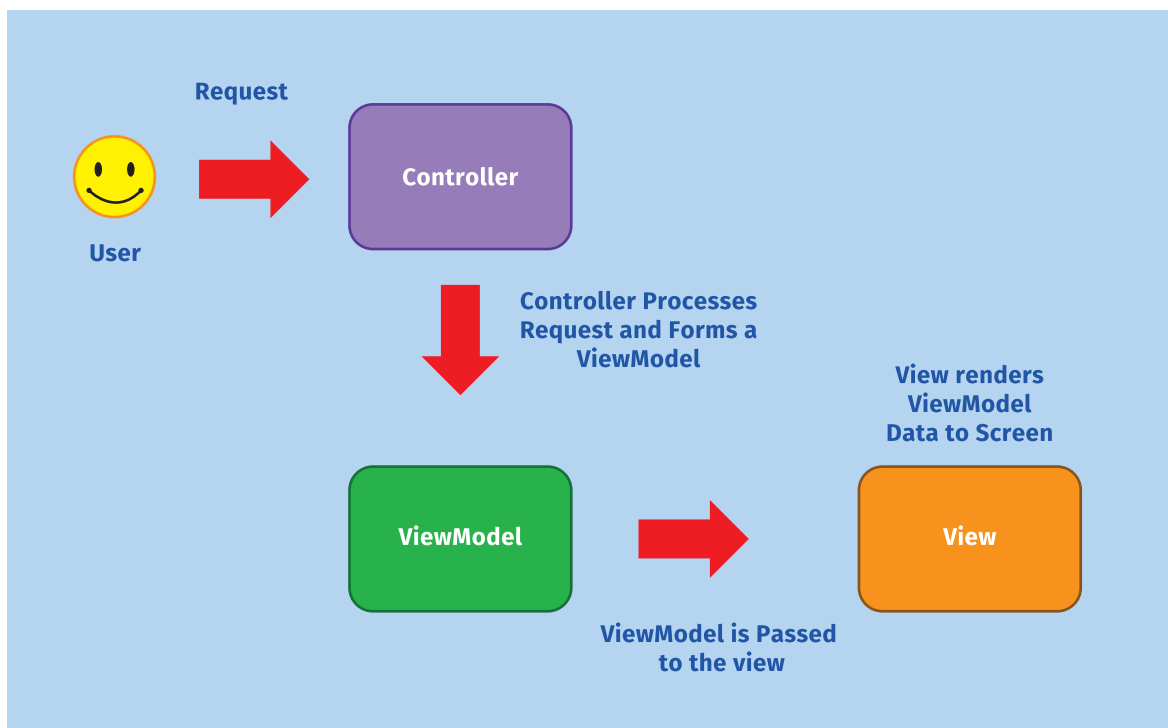


Figure 2: Request Flow

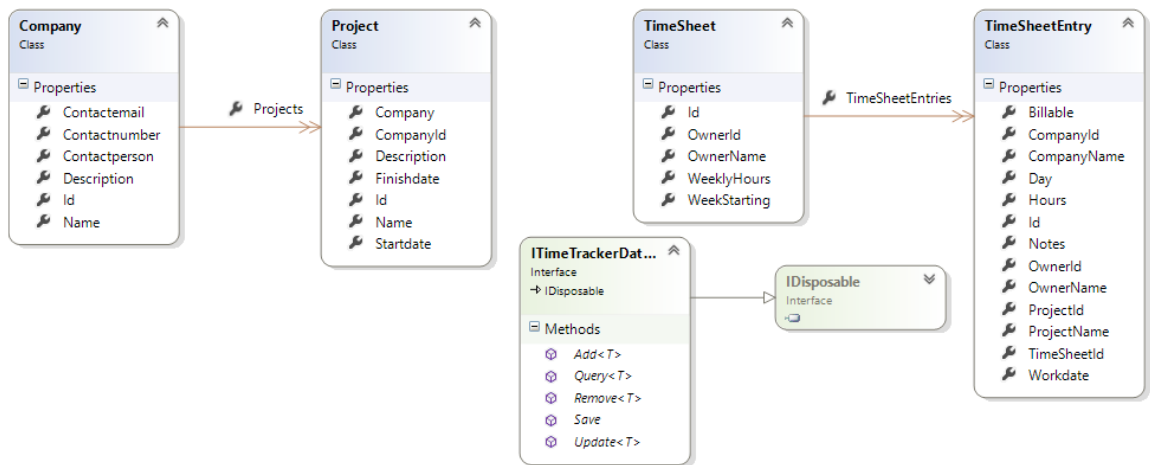


Figure 3: timetracker.Domain Class Diagram

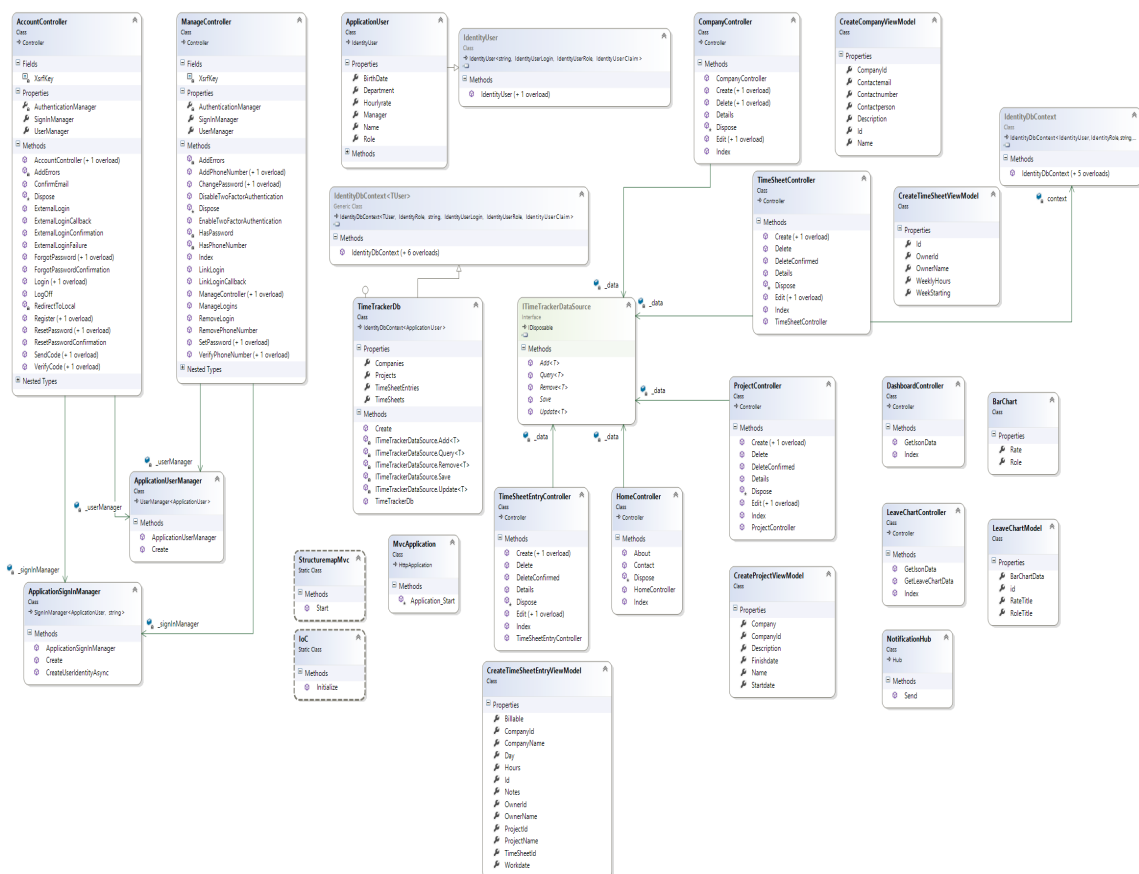


Figure 4: timetracker.Web Class Diagram

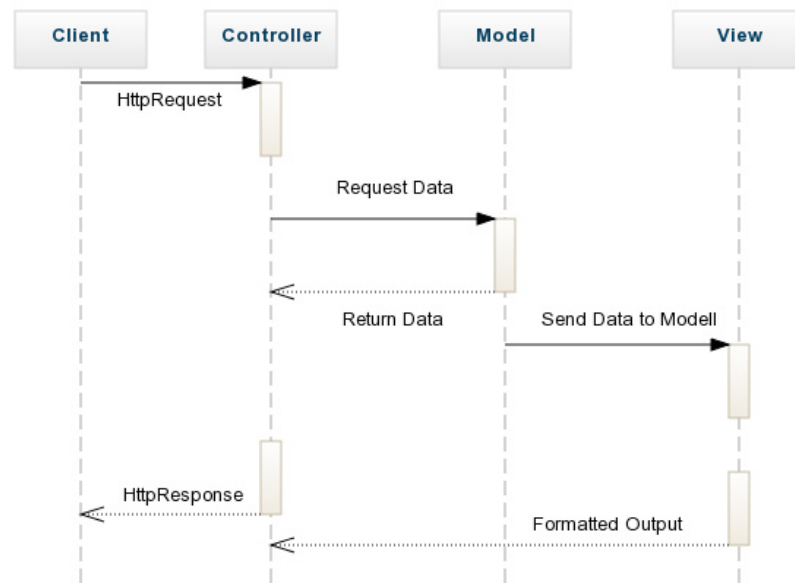


Figure 5: Request Sequence

Problems

Issue: Data Access

Data access in the TimeTracker application was abstracted using an interface to access the domain entities. While this achieved the aim of decoupling the controllers from the database context class TimeTrackerDb it did present a problem when a controller needed to perform an action on a domain entity. As the controller classes should not know anything about TimeTrackerDb hard coding in a dependency was avoided. Instead each controller had a constructor that accepted an instance of the ITimeTrackerDataSource interface and assigned it to a private field but the MVC runtime did not know how to create a controller without a default constructor. When the code was run an error was generated stating, **“No parameterless constructor defined for this object”**. So the solution was to use Dependency Injection to introduce the TimeTrackerDb to the controllers. This was implemented using an Inversion of Control container called StructureMap.. StructureMap adds a file to the project that was used to configure the MVC runtime to pass in a concrete instance of TimeTrackerDb every time a constructor requiring an instance of the ITimeTrackerDataSource interface was called.

Issue: ViewModels being saved to Database

In order to preserve loose coupling within TimeTracker View Models were used. View Models represent the data from the domain model that needs to be displayed to the user. This removes any need for UI logic in the View and means the only responsibility or concern of the View is rendering that View Model to the screen. This helps to ensure the application adheres to the principle of separation of concerns that underpins the MVC architecture. This helped to make the application more organised and testable but also threw up a problem. When Entity framework created the database it was also adding tables for these View Models. After researching the issue I realised the problem was caused by scaffolding in the MVC project. Scaffolding in MVC 5 is a code generation framework that allows developers to quickly add code that interacts with data models. When I created my Views I used scaffolding to create a strongly typed View using the View Model as the model, this then added a DbSet to my TimeTrackerDb. When Entity created the database it inspected this class and generated the database schema based on it and therefore was adding View Models to the database. To avoid this these Views were deleted and new ones were recreated that were not strongly typed, and the View Models were no longer being created in the database schema.

Issue: Creating a Single Database Context

The ASP.NET Identity Framework was used to handle user membership and authorization within the TimeTracker application. By default Identity stores the user information in a database through a class called ApplicationDbContext. This class uses a connection string called “DefaultConnection” that is configured in the web.config file. This connection string refers to a SQL local DB database. As the application was using a separate database context TimeTrackerDB for application data, a problem occurred when it was necessary to associate a user with another entity. Initially TimeTrackerDb inherited from the Entity Framework class DbContext . To allow all users and application data to be stored in the same database it was necessary to inherit from IdentityDbContext<ApplicationUser>. This class itself inherits from DbContext but also includes Dbsets for Users and Roles. A default constructor was also created in TimeTrackerDb that included the connection string “DefaultConnection” so that both the user database context and the application database context were now pointing to the same database. This involved removing the initial database and recreating it based on the new configuration

Testing

Unit testing

Unit testing was carried out using the built in Microsoft unit test framework that is available in Visual Studio. Visual Studio also provides Test Explorer, which allows tests to be run from within the same solution as the project. Test results can then be viewed and debugged without leaving the Visual Studio environment. This provides an integrated solution environment that promotes Test Driven Development.

Where possible TDD was used to drive the design and development of the project. TDD involves writing test cases before the functionality to be tested is created. It is intended to help ensure functionality is clearly thought out and the resulting software is flexible and robust. TDD follows the Red-Green-Refactor pattern.

Red – write a failing test

Green – do the simplest thing to make the test pass

Refactor – refactor the code to improve the design

Using this method also means that a suite of test cases are generated that can be used any time changes are made in order to ensure that the system still functions correctly.

The MVC pattern also makes testing easier as the business logic and user interface logic are separated into components. This allows controllers to be tested independently of the model and the view that renders the UI. In TimeTracker the database is only accessed through an interface therefore the controllers are not tightly coupled to the database. This meant that a “Fake” database context could be created that implemented this interface as part of the test suite. This ‘Fake” database was then passed in to controllers under test, allowing the controllers to be tested without requiring access to the real database. Fake test data was also created and stored in memory and used when running these tests. This method ensures that running test cases was quick allowing tests to be run frequently to ensure the units all worked correctly.

Unit test were carried out using the Arrange-Act-Assert structure.

- Arrange – Setup the unit to be tested.
- Act – Execute the unit to be tested and collect results.
- Assert – Verify the collected results.


```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using timeTracker.Domain;
7
8  namespace timeTracker.Web.Tests.Fakes
9  {
10     class FakeTimeTrackerDb : ITimeTrackerDataSource
11     {
12         public IQueryable<T> Query<T>() where T : class
13         {
14             return Sets[typeof(T)] as IQueryable<T>;
15         }
16
17         public void Dispose() { }
18
19         public void AddSet<T>(IQueryable<T> objects)
20         {
21             Sets.Add(typeof(T), objects);
22         }
23
24         public void Add<T>(T entity) where T : class
25         {
26             Added.Add(entity);
27         }
28
29         public void Update<T>(T entity) where T : class
30         {
31             Updated.Add(entity);
32         }
33
34         public void Remove<T>(T entity) where T : class
35         {
36             Removed.Add(entity);
37         }
38
39         public void Save()
40         {
41             Saved = true;
42         }
43     }
44
45     public Dictionary<Type, object> Sets = new Dictionary<Type, object>();
46     public List<object> Added = new List<object>();
47     public List<object> Updated = new List<object>();
48     public List<object> Removed = new List<object>();
49     public bool Saved = false;
50 }
51
52

```

Figure 6: Fake Database Context for testing

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using timeTracker.Web.ViewModels;
7  using timeTracker.Web.Models;
8  using timeTracker.Domain;
9
10
11 namespace timeTracker.Web.Tests
12 {
13     class TestData
14     {
15         public static IQueryable<Company> Companies
16         {
17             get
18             {
19                 var companies = new List<Company>();
20
21                 for (int i = 0; i < 100; i++)
22                 {
23                     var company = new Company();
24                     company.Id = i;
25                     company.Projects = new List<Project>(){
26                         new Project { Id = i}
27                     };
28                     companies.Add(company);
29                 }
30                 return companies.AsQueryable();
31             }
32         }
33     }
34 }
35

```

Figure 7: Test Data for testing Company Controller

```

1  using System;
2  using Microsoft.VisualStudio.TestTools.UnitTesting;
3  using timeTracker.Web.Controllers;
4  using System.Web.Mvc;
5  using timeTracker.Web.Tests.Fakes;
6  using timeTracker.Web.Models;
7  using timeTracker.Web.ViewModels;
8  using timeTracker.Domain;
9  using System.Linq;
10
11
12 namespace timeTracker.Web.Tests.Controllers
13 {
14     [TestClass]
15     public class CompanyControllerTest
16     {
17
18         [TestMethod]
19         public void Create_Saves_Company_When_Valid()
20         {
21             // Arrange
22             var db = new FakeTimeTrackerDb();
23             var controller = new CompanyController(db);
24
25             // Act
26             controller.Create(new CreateCompanyViewModel());
27
28             //Assert
29             Assert.AreEqual(1, db.Added.Count);
30             Assert.AreEqual(true, db.Saved);
31         }
32
33         [TestMethod]
34         public void Create_Does_not_Save_Company_When_Invalid()
35         {
36             // Arrange
37             var db = new FakeTimeTrackerDb();
38             var controller = new CompanyController(db);
39
40             //Act
41             controller.ModelState.AddModelError("", "Invalid");
42             controller.Create(new CreateCompanyViewModel());
43
44             //Assert
45             Assert.AreEqual(0, db.Added.Count);
46         }
47
48         [TestMethod]
49         public void Details_Is_Not_Null()
50         {
51             // Arrange
52             var db = new FakeTimeTrackerDb();
53             db.AddSet(TestData.Companies);
54             var controller = new CompanyController(db);
55
56             //Act
57             ViewResult result = controller.Details(1) as ViewResult;
58
59             //Assert
60             Assert.IsNotNull(result);
61         }
62
63         [TestMethod]
64         public void Edit_Saves_Company_When_Valid(){
65             // Arrange
66             var db = new FakeTimeTrackerDb();
67             db.AddSet(TestData.Companies);
68             var controller = new CompanyController(db);
69             var company = db.Query<Company>().Single(c => c.Id == 1);
70
71             // Act
72             controller.Edit(company);
73
74             //Assert
75             Assert.AreEqual(1, db.Updated.Count);
76             Assert.AreEqual(true, db.Saved);
77         }
78     }
79
80     [TestMethod]
81     public void Edit_Saves_Company_When_Invalid()
82     {
83         // Arrange
84         var db = new FakeTimeTrackerDb();
85         db.AddSet(TestData.Companies);
86         var controller = new CompanyController(db);
87         var company = db.Query<Company>().Single(c => c.Id == 1);
88
89         //Act
90         controller.ModelState.AddModelError("", "Invalid");
91         controller.Edit(company);
92
93         //Assert
94         Assert.AreEqual(0, db.Updated.Count);
95     }
96 }
97

```

Figure 8: Company Controller Test Suite

Automated testing

Manually testing a web application can be a time consuming process as tests need to be completed in sequence and on a complete system. These tests also need to be redone after every major change to a module or component to ensure the system still works correctly. With this in mind Selenium IDE was chosen to automate integration testing. Selenium IDE is a web testing plugin for Mozilla Firefox that allows for the creation and automation of test cases. To set up automated testing I configured the IDE to work with the URL of my application I then made a series of recordings of my browser interactions. These recordings were then used to generate a suite of automated test cases that could then be automated to run when needed.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
4 <head profile="http://selenium-ide.openqa.org/profiles/test-case">
5 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
6 <link rel="selenium.base" href="http://localhost:57011/" />
7 <title>Admin_Login_Valid</title>
8 </head>
9 <body>
10 <table cellpadding="1" cellspacing="1" border="1">
11 <thead>
12 <tr><td rowspan="1" colspan="3">Admin_Login_Valid</td></tr>
13 </thead><tbody>
14 <tr>
15 <td>open</td>
16 <td>/</td>
17 <td></td>
18 </tr>
19 <tr>
20 <td>clickAndWait</td>
21 <td>id=loginLink</td>
22 <td></td>
23 </tr>
24 <tr>
25 <td>type</td>
26 <td>id=Email</td>
27 <td>admin@mail.com</td>
28 </tr>
29 <tr>
30 <td>type</td>
31 <td>id=Password</td>
32 <td>ChangeItAsap0!</td>
33 </tr>
34 <tr>
35 <td>clickAndWait</td>
36 <td>id=loginBtn</td>
37 <td></td>
38 </tr>
39 </tbody></table>
40 </body>
41 </html>
42
43
```

Figure 9: Test Case to check valid admin login

User Validation

Employees at Spanish Point Technologies Ltd will provide user feedback on the TimeTracker web application at a time convenient to them before project demo. It is hoped that this will highlight any issues that might occur in a production environment. This feedback will then be used to fix any bugs that occur in the software in real life situation. Requests or suggestions for new features will then be researched and considered for implementation in the future.

Future work

Scaling and Performance

TimeTracker was intended for use by small to medium enterprises. In the future to scale the application to deal with larger enterprises and the increased complexity the application would be reassessed to evaluate what could be refactored to avoid bottlenecks caused by;

- Database Access
- Bandwidth Issues
- Processing Power

Currently the number of concurrent threads allowable is set to 5000 by default.

One way to increase this for larger scale application would be to use Asynchronous controllers to allow long running requests to run asynchronously.

Data Analysis and Prediction

As an application scales the amount of data being created also scales and becomes increasingly complex to use efficiently. This data can be a very valuable tool to a business if used correctly. In the future I would like to add features to TimeTracker that would allow this data to be mined and used for predictive analysis. For example, predicting the estimated completion date of a project based upon previous similar projects.

Appendices

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace timeTracker.Domain
{
    public interface ITimeTrackerDataSource : IDisposable
    {

        IQueryable<T> Query<T>() where T : class;
        void Add<T>(T entity) where T : class;
        void Update<T>(T entity) where T : class;
        void Remove<T>(T entity) where T : class;
        void Save();

    }
}
```

```

g System;
g System.Collections.Generic;
g System.ComponentModel.DataAnnotations;
g System.Linq;
g System.Text;
g System.Threading.Tasks;

space timeTracker.Domain

public class Company
{
    public virtual int Id { get; set; }
    public virtual string Name { get; set; }

    [Display(Name = "Contact Person")]
    public virtual string Contactperson { get; set; }

    [Display(Name = "Contact Number")]
    public virtual string Contactnumber { get; set; }

    [Display(Name = "Contact Email")]
    public virtual string Contactemail { get; set; }

    public virtual string Description { get; set; }

    // Collection to hold related projects
    public virtual ICollection<Project> Projects { get; set; }
}

```

```

ig System;
ig System.Collections.Generic;
ig System.ComponentModel.DataAnnotations;
ig System.ComponentModel.DataAnnotations.Schema;
ig System.Linq;
ig System.Text;
ig System.Threading.Tasks;

:space timeTracker.Domain

public class Project
{
    public virtual int Id { get; set; }
    public virtual string Name { get; set; }
    public virtual string Company { get; set; }
    public virtual int CompanyId { get; set; }
    public virtual string Description { get; set; }

    [DataType(DataType.Date)]
    [Display(Name = "Start Date")]
    public virtual DateTime? Startdate { get; set; }

    [DataType(DataType.Date)]
    [Display(Name = "Finish Date")]
    public virtual DateTime? Finishdate { get; set; }
}

```

```

ig System;
ig System.Collections.Generic;
ig System.ComponentModel.DataAnnotations;
ig System.ComponentModel.DataAnnotations.Schema;
ig System.Linq;
ig System.Text;
ig System.Threading.Tasks;

:space timeTracker.Domain

public class TimeSheet
{
    public virtual int Id { get; set; }
    public virtual string OwnerId { get; set; }

    [Display(Name = "Employee")]
    public virtual string OwnerName { get; set; }

    [DataType(DataType.Date)]
    [Display(Name = "Week Starting")]
    public virtual DateTime? WeekStarting { get; set; }

    [Display(Name = "Hours")]
    public virtual float WeeklyHours { get; set; }

    // Collection to hold daily timesheet entries
    [Display(Name = "Entries")]
    public virtual ICollection<TimeSheetEntry> TimeSheetEntries { get; set; }
}

```



```

ig System;
ig System.Collections.Generic;
ig System.ComponentModel.DataAnnotations;
ig System.ComponentModel.DataAnnotations.Schema;
ig System.Linq;
ig System.Text;
ig System.Threading.Tasks;

:space timeTracker.Domain

public class TimeSheetEntry
{
    public virtual int Id { get; set; }
    public virtual int TimeSheetId { get; set; }
    public virtual string OwnerId { get; set; }
    public virtual int CompanyId { get; set; }
    public virtual int ProjectId { get; set; }
    public virtual string Notes { get; set; }
    public virtual bool Billable { get; set; }
    public virtual string Day { get; set; }
    public virtual int Hours { get; set; }

    [Display(Name = "Employee")]
    public virtual string OwnerName { get; set; }

    [Display(Name = "Company")]
    public virtual string CompanyName { get; set; }

    [Display(Name = "Project")]
    public virtual string ProjectName { get; set; }

    [DataType(DataType.Date)]
    [Display(Name = "Date")]
    public virtual DateTime? Workdate { get; set; }
}

```

```

ig Microsoft.AspNet.Identity.EntityFramework;
ig System;
ig System.Collections.Generic;
ig System.Data.Entity;
ig System.Linq;
ig System.Web;
ig timeTracker.Domain;
ig timeTracker.Web.Models;

:space timeTracker.Web.Infrastructure

// Database context class inherits from IdentityDbContext<ApplicationUser>
// which handles users
// Implements ITimeTrackerDataSource
public class TimeTrackerDb : IdentityDbContext<ApplicationUser>,
ITimeTrackerDataSource
{
    // Database Connection
    public TimeTrackerDb() : base("DefaultConnection") { }

    public static TimeTrackerDb Create()
    {
        return new TimeTrackerDb();
    }

    // Entity sets for create, read, update and delete
    public DbSet<Project> Projects { get; set; }
    public DbSet<Company> Companies { get; set; }
    public DbSet<TimeSheet> TimeSheets { get; set; }
    public DbSet<TimeSheetEntry> TimeSheetEntries { get; set; }

    // Read
    // Returns a set of queryable objects
    IQueryable<T> ITimeTrackerDataSource.Query<T>()
    {
        return Set<T>();
    }

    // Create
    void ITimeTrackerDataSource.Add<T>(T entity)
    {
        Set<T>().Add(entity);
    }

    // Delete
    void ITimeTrackerDataSource.Remove<T>(T entity)
    {
        Set<T>().Remove(entity);
    }

    //Update
    void ITimeTrackerDataSource.Update<T>(T entity)
    {

```

```
        Entry(entity).State = System.Data.Entity.EntityState.Modified;
    }

    // Save changes to database
    void ITimeTrackerDataSource.Save()
    {
        SaveChanges();
    }
}
```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Web;
using System.Web.Mvc;
using timeTracker.Domain;
using timeTracker.Web.ViewModels;
using PagedList;

namespace timeTracker.Web.Controllers
{
    public class ProjectController : Controller
    {
        //Datasource Interface
        private readonly ITimeTrackerDataSource _data;

        //Constructor that takes a parameter. Concrete implementation passed in a
        //time using IOC
        public ProjectController(ITimeTrackerDataSource data)
        {
            _data = data;
        }

        // GET: Project
        //Returns an Index page of all Projects with search box, ordering and
        //ng.
        public ActionResult Index(string sortOrder, string currentFilter, string
        searchString, int? page)
        {
            ViewBag.CurrentSort = sortOrder;
            ViewBag.ProjectNameSortParam = String.IsNullOrEmpty(sortOrder) ?
            "project_desc" : "";
            ViewBag.CompanyNameSortParam = String.IsNullOrEmpty(sortOrder) ?
            "company_desc" : "";
            ViewBag.StartDateSort = String.IsNullOrEmpty(sortOrder) ? "start_desc
            " : "";
            ViewBag.FinishDateSort = String.IsNullOrEmpty(sortOrder) ?
            "finish_desc" : "";

            if (searchString != null)
            {
                page = 1;
            }
            else
            {
                searchString = currentFilter;
            }

            ViewBag.CurrentFilter = searchString;
            var projects = _data.Query<Project>();

            if (!String.IsNullOrEmpty(searchString))
            {

```

```

        projects = _data.Query<Project>().Where(c =>
            c.Name.Contains(searchString));
    }

    switch (sortOrder)
    {
        case "project_desc":
            projects = projects.OrderByDescending(p => p.Name);
            break;

        case "company_desc":
            projects = projects.OrderByDescending(p => p.Company);
            break;

        case "start_desc":
            projects = projects.OrderByDescending(p => p.Startdate);
            break;

        case "finish_desc":
            projects = projects.OrderByDescending(p => p.Finishdate);
            break;

        default:
            projects = projects.OrderBy(p => p.Name);
            break;
    }

    int pageSize = 4;
    int pageNumber = (page ?? 1);

    return View(projects.ToPagedList(pageNumber, pageSize));
}

// GET: Project/Details/5
//Returns Details of a Project given an Id passed as parameter
public ActionResult Details(int id)
{
    var model = _data.Query<Project>().Single(d => d.Id == id);
    return View(model);
}

// GET: Project/Create
// Returns a form for creating a new Project
[HttpGet]
public ActionResult Create(int companyId, string companyName)
{
    var model = new CreateProjectViewModel();
    model.CompanyId = companyId;
    model.Company = companyName;
    return View(model);
}

// POST: Project/Create
// Persists newly created Project to database after a validity check
[HttpPost]
public ActionResult Create(CreateProjectViewModel viewModel)

```

```

{
    if(ModelState.IsValid)
    {
        //Get the related Company
        var company = _data.Query<Company>().Single(c => c.Name ==
Model.Company);

        //Create a new project
        var project = new Project
        {
            Name = viewModel.Name,
            Company = viewModel.Company,
            CompanyId = company.Id,
            Description = viewModel.Description,
            Startdate = viewModel.Startdate,
            Finishdate = viewModel.Finishdate
        };

        //Add it to the Company's list of projects
        company.Projects.Add(project);
        _data.Add(project);
        _data.Save();
        return RedirectToAction("details", "company", new {id =
Model.CompanyId});
    }
    return View(viewModel);
}

// GET: Project/Edit/5
// Modifies Project in database after validity check
[HttpGet]
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    var project = _data.Query<Project>().Single(t => t.Id == id);
    if (project == null)
    {
        return HttpNotFound();
    }
    return View(project);
}

// POST: PROJECT/Edit/5
// Returns a form to edit a Project's details
// Restricted to Admin
[HttpPost, ActionName("Edit")]
[Authorize(Roles = "Admin")]
[ValidateAntiForgeryToken]
public ActionResult Edit(Project project)
{
    if (ModelState.IsValid)
    {

```

```

        _data.Update(project);
        _data.Save();
        return RedirectToAction("Index");
    }
    return View(project);
}

// GET: Project/Delete/5
// Returns details page of the project to be deleted
// Restricted to Admin
[HttpGet]
[Authorize(Roles = "Admin")]
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }

    var project = _data.Query<Project>().Single(t => t.Id == id);

    if (project == null)
    {
        return HttpNotFound();
    }
    return View(project);
}

// POST: Project/Delete/5
// Removes Project from database and returns page confirming deletion
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(int id)
{
    var project = _data.Query<Project>().Single(t => t.Id == id);
    var company = _data.Query<Company>().Single(d => d.Id ==
ect.CompanyId);
    _data.Remove(project);
    _data.Save();
    return RedirectToAction("details", "company", new { id =
ect.CompanyId });
}

protected override void Dispose(bool disposing)
{
    _data.Dispose();
    base.Dispose(disposing);
}

}

```

```

ig System;
ig System.Collections.Generic;
ig System.Linq;
ig System.Net;
ig System.Web;
ig System.Web.Mvc;
ig timeTracker.Domain;
ig timeTracker.Web.Infrastructure;
ig timeTracker.Web.ViewModels;
ig PagedList;

```

```

:space timeTracker.Web.Controllers

```

```

public class CompanyController : Controller
{

    //Datasource Interface
    private readonly ITimeTrackerDataSource _data;

    //Constructor that takes a parameter. Concrete implementation passed in a
    time using IoC
    public CompanyController(ITimeTrackerDataSource data)

    {
        _data = data;
    }

    //GET: Company
    //Returns an Index page of all Companies with search box, ordering and
    ng.
    public ActionResult Index(string sortOrder, string currentFilter, string
    chString, int? page)
    {
        ViewBag.CurrentSort = sortOrder;
        ViewBag.NameSortParam = String.IsNullOrEmpty(sortOrder) ? "name_desc"

        ViewBag.ContactNameSortParam = String.IsNullOrEmpty(sortOrder) ?
    tact_desc" : "";

        if (searchString != null)
        {
            page = 1;
        }
        else
        {
            searchString = currentFilter;
        }

        ViewBag.CurrentFilter = searchString;

        var companies = _data.Query<Company>();

        if (!String.IsNullOrEmpty(searchString))

```



```

        {
            companies = _data.Query<Company>().Where(c =>
me.Contains(searchString));
        }

        switch (sortOrder)
        {
            case "name_desc":
                companies = companies.OrderByDescending(c => c.Name);
                break;

            case "contact_desc":
                companies = companies.OrderByDescending(c => c.Contactperson)
                break;

            default:
                companies = companies.OrderBy(c => c.Name);
                break;
        }

        int pageSize = 4;
        int pageNumber = (page ?? 1);

        return View(companies.ToPagedList(pageNumber, pageSize));
    }

    // GET: Company/Details/5
    // Returns Details of a Company given an Id passed as parameter
    public ActionResult Details(int id)
    {
        var model = _data.Query<Company>().Single(d => d.Id == id);

        if (model == null)
        {
            return HttpNotFound();
        }

        return View(model);
    }

    // GET: Company/Create
    // Returns a form for creating a new Company
    // Restricted to Admin
    [Authorize(Roles="Admin")]
    [HttpGet]
    public ActionResult Create()
    {
        return View();
    }

    // POST: Company/Create
    // Persists newly created Company to database after a validity check
    [HttpPost]

```

```

public ActionResult Create(CreateCompanyViewModel viewModel)
{
    if (ModelState.IsValid)
    {
        var company = new Company
        {
            Name = viewModel.Name,
            Contactperson = viewModel.Contactperson,
            Contactnumber = viewModel.Contactemail,
            Contactemail = viewModel.Contactemail,
            Description = viewModel.Description
        };

        _data.Add(company);
        _data.Save();

        return RedirectToAction("Index", "Company");
    }

    return View(viewModel);
}

// GET: Company/Edit/5
// Returns a form to edit a Company's details
// Restricted to Admin
[Authorize(Roles = "Admin")]
[ValidateAntiForgeryToken]
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }

    var company = _data.Query<Company>().Single(c => c.Id == id);

    if (company == null)
    {
        return HttpNotFound();
    }

    return View(company);
}

// POST: Company/Edit/5
// Modifies Company in database after validity check
[HttpPost, ActionName("Edit")]
[ValidateAntiForgeryToken]
public ActionResult Edit(Company company)
{
    if (ModelState.IsValid)
    {
        _data.Update(company);
        _data.Save();
        return RedirectToAction("Index");
    }
}

```

```

        return View(company);
    }

    // GET: Company/Delete/5
    // Returns a page with details of Company to be deleted
    //Restricted to Admin
    [HttpGet]
    [Authorize(Roles = "Admin")]
    public ActionResult Delete(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
        var company = _data.Query<Company>().Single(t => t.Id == id);
        if (company == null)
        {
            return HttpNotFound();
        }
        return View(company);
    }

    // POST: Company/Delete/5
    // Removes Company from database and returns page confirming deletion
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public ActionResult DeleteConfirmed(int id)
    {
        var company = _data.Query<Company>().Single(t => t.Id == id);
        _data.Remove(company);
        _data.Save();
        return RedirectToAction("Index");
    }

    protected override void Dispose(bool disposing)
    {
        _data.Dispose();
        base.Dispose(disposing);
    }
}

```

Supervision Meeting

Date: 3 / 3 / 2015

Student(s): Aoibhinn Farrell

Project/Practicum:

Supervisor: Rosalind Verbruggen

Progress since previous meeting

Dataman + UI working

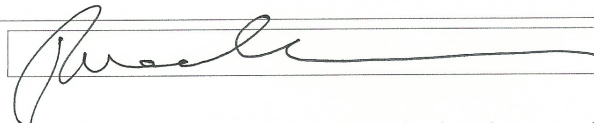
Planned Activities

Update to reports
eg. Schedule of the future
Cost per hour etc.

Supervisor's Comments

Good progress

Supervisor's Signature



Students must retain these signed forms and submit them to the project/practicum coordinator in order to be entitled to present their project/practicum.

Supervision Meeting

Date: 30/3/15

Student(s): Aoibhinn Farrell

Project/Practicum: TimeTracker - ASP.NET Web Application

Supervisor: Ruwaa Verbruggen

Progress since previous meeting

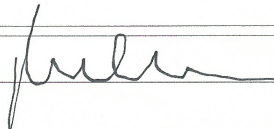
CRUD functionality completed
Real time Notifications added

Planned Activities

Add Interactive graphs

Supervisor's Comments

Supervisor's Signature



Student's must retain these signed forms and submit them to the project/practicum coordinator in order to be entitled to present their project/practicum.

Supervision Meeting

Date: 14/4/2015

Student(s): Aoibhinn Farrell

Project/Practicum: TimeTracker - ASP.NET Web Application

Supervisor: Robert Verbruggen

Progress since previous meeting

Continuing on GitHub

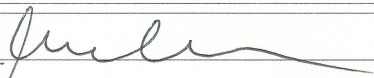
Planned Activities

Complete GitHub account

Supervisor's Comments

Good progress

Supervisor's Signature



Student's must retain these signed forms and submit them to the project/practicum coordinator in order to be entitled to present their project/practicum.

Supervision Meeting

Date: 30/3/15

Student(s): Aoibhinn Farrell

Project/Practicum: TimeTracker - ASP.NET Web Application

Supervisor: Ruwaa Verbruggen

Progress since previous meeting

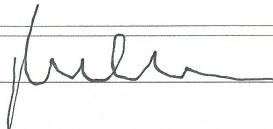
CRUD functionality completed
Real time Notifications added

Planned Activities

Add interactive graphs

Supervisor's Comments

Supervisor's Signature



Student's must retain these signed forms and submit them to the project/practicum coordinator in order to be entitled to present their project/practicum.